

Máster Avanzado de Programación en Python para Hacking, Big Data y Machine Learning

DESARROLLO SEGURO EN PYTHON

LECCIÓN 03

Controles Proactivos OWASP TOP 10

ÍNDICE

Introducción

Análisis

Objetivos

Conclusiones

INTRODUCCIÓN

El TOP 10 de controles proactivos de OWASP es una lista de técnicas de seguridad que deben considerarse para cada proyecto de desarrollo software seguro.

Uno de los principales objetivos de esta lista es proporcionar una guía práctica concreta que ayude a los desarrolladores a crear software seguro. Estas técnicas deben aplicarse de forma proactiva en las primeras etapas del desarrollo de software para garantizar la máxima eficacia.

OBJETIVOS

Al finalizar esta lección serás capaz de:

- 1 Conocer que son los controles proactivos OWASP
- 2 Aplicar los controles proactivos en el desarrollo de nuestras aplicaciones
- 3 Mejorar la seguridad en el desarrollo software
- 4 Ampliar la visión de la seguridad en el desarrollo de aplicaciones

CONTROLES PROACTIVOS

Lista de controles proactivos en desarrollo

- C1: Definir requisitos de seguridad
- C2: Aprovechar las bibliotecas y los frameworks de seguridad
- C3: Acceso seguro a la base de datos
- C4: Codificar y escapar datos
- C5: Validar todas las entradas
- C6: Implementar la identidad digital
- C7: Aplicar controles de acceso
- C8: Proteja los datos en todas partes
- C9: Implementar el registro y la supervisión de seguridad
- C10: Manejar todos los errores y excepciones



C1: Definir requisitos de seguridad

Los requisitos de seguridad se derivan de:

- los estándares de la industria.
- las leyes aplicables.
- un historial de vulnerabilidades pasadas.

Los requisitos de seguridad definen nuevas funciones o añaden características a las existentes para resolver un problema de seguridad específico o eliminar una vulnerabilidad potencial.



C1: Definir requisitos de seguridad

Reutilizar controles de seguridad

En lugar de crear un enfoque de seguridad personalizado para cada aplicación, los requisitos de seguridad estándar permiten a los desarrolladores reutilizar la definición de controles de seguridad y mejores prácticas. Esos mismos requisitos de seguridad examinados proporcionan soluciones para problemas de seguridad que se han producido en el pasado. Existen requisitos para evitar la repetición de fallos de seguridad pasadas.



C1: Definir requisitos de seguridad

OWASP ASVS

OWASP Application Security Verification Standard (ASVS) es un catálogo de requisitos de seguridad y criterios de verificación disponibles. Puede ser una fuente de requisitos de seguridad detallados para los equipos de desarrollo.



Aumento de requisitos con historias de usuarios y casos de uso indebido



Una historia de usuario o un caso de uso indebido se vincula la aplicación exactamente con lo que el usuario o atacante hace con el sistema, en lugar de describir lo que el sistema ofrece al usuario.

C1: Definir requisitos de seguridad


Ejemplo de aplicación de un requisito de ASVS

De la sección "Requisitos de verificación de autenticación" de ASVS 3.0.1, el requisito 2.19 se centra en las contraseñas predeterminadas.

2.19 Verifique que no haya contraseñas predeterminadas en uso para el marco de la aplicación o cualquier componente utilizado por la aplicación (como "admin / password").

Cuando la historia se centra en el atacante y sus acciones, se denomina **caso de uso indebido**.

Como atacante, puedo ingresar un nombre de usuario y contraseña predeterminados para obtener acceso.

- 
- Implementación
 - Descubrimiento y selección
 - Investigación y documentación
 - Implementación y prueba

C1: Definir requisitos de seguridad

Vulnerabilidades evitadas

Los requisitos de seguridad definen la funcionalidad de seguridad de una aplicación. **Una mejor seguridad incorporada desde el comienzo del ciclo de vida de las aplicaciones da como resultado la prevención de muchos tipos de vulnerabilidades.**

```
function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
      if (a) {
        (; o > i; i++)
          if (r = t.call(e[i], i, e[i]), r === !1) break
      }
      (i in e)
        if (r = t.call(e[i], i, e[i]), r === !1) break;
      e
    }
  )
  b.call("\uffeff\u00a0") ? function(e) {
    null == e ? "" : b.call(e)
  } : function(e) {
    null == e ? "" : (e + "").replace(C, "")
  }
  function(e, t) {
    t || [];
    null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : n)
  }
  function(e, t, n) {
    (
      (a) return m.call(t, e, n);
      (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
      if (n in t && t[n] === e) return n
    )
  }
}
```

C2: Aprovechar las bibliotecas y los frameworks de seguridad

Las librerías de codificación seguras y los frameworks con seguridad integrada ayudan a los desarrolladores de software a protegerse contra los fallos de implementación y diseño relacionados con la seguridad.

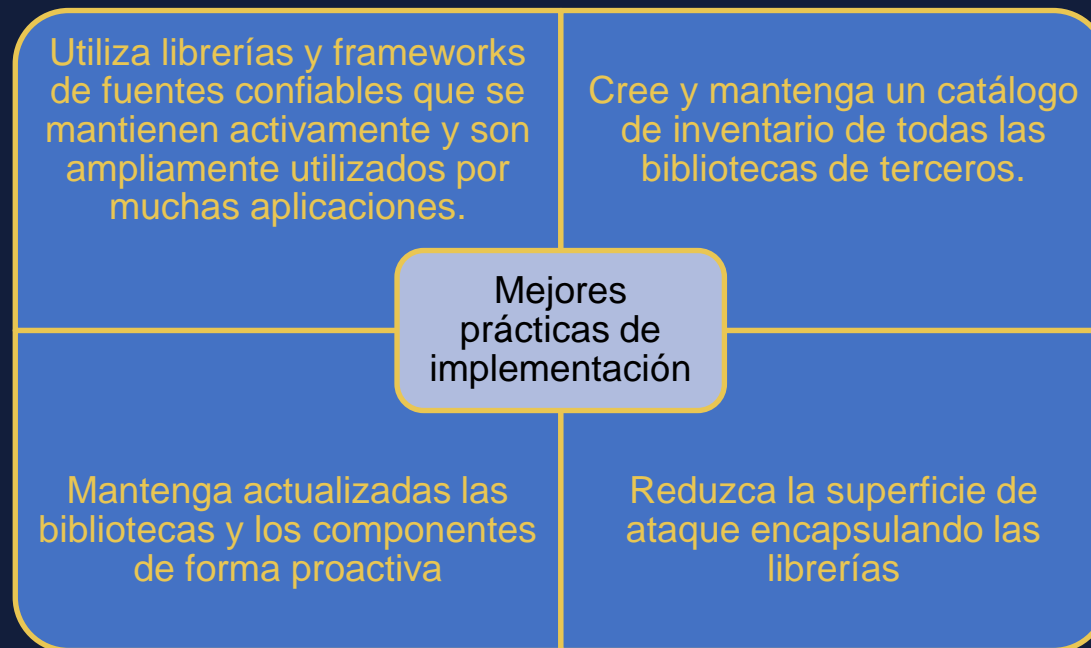
Un desarrollador que escribe una aplicación desde cero puede no tener el conocimiento, el tiempo o el presupuesto suficiente para implementar o mantener las características de seguridad de manera adecuada.



C2: Aprovechar las bibliotecas y los frameworks de seguridad

Mejores prácticas de implementación

Al incorporar librerías o frameworks de terceros en tu software, es importante tener en cuenta las siguientes buenas prácticas.



C2: Aprovechar las bibliotecas y los frameworks de seguridad

Vulnerabilidades evitadas

Los frameworks y librerías seguras pueden ayudar a prevenir un amplio rango de vulnerabilidades en aplicaciones web. Es fundamental mantener estos frameworks y librerías actualizadas.

Herramientas

[OWASP Dependency-Check](#)

[Retire.js](#)

```
function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
    }
    else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break
    if (a) {
      (; o > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
    }
    (i in e)
      if (r = t.call(e[i], i, e[i]), r === !1) break;
    e

    !b.call("\uffeff\u00a0") ? function(e) {
      null == e ? "" : b.call(e)
    }
    n(e) {
      null == e ? "" : (e + "").replace(C, "")
    }

    function(e, t) {
      t || [];
      null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
    )
    function(e, t, n) {
      (
        (a) return m.call(t, e, n);
        (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
        if (n in t && t[n] === e) return n
      )
    }
  )
}
```


C3: Acceso seguro a la base de datos

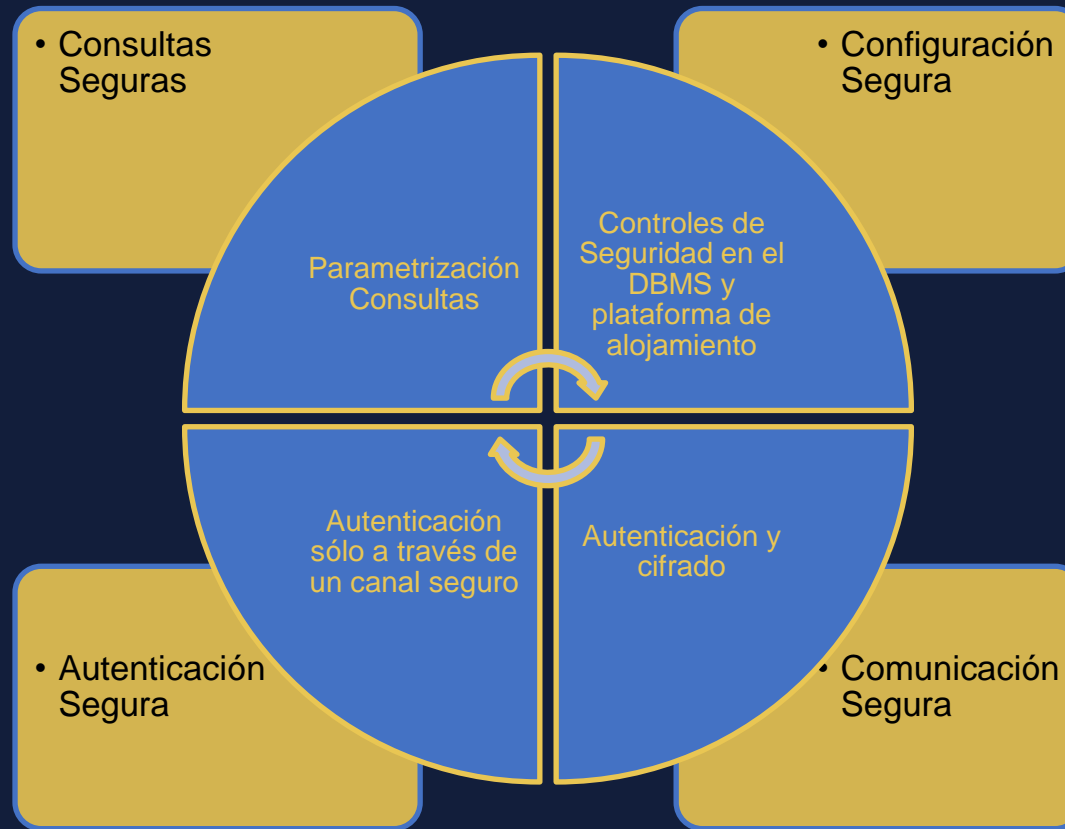
Acceso seguro a todos los almacenes de datos, incluidas las bases de datos relacionales y las bases de datos NoSQL.

Algunas áreas para considerar:

- Consultas seguras
- Configuración segura
- Autenticación segura
- Comunicación segura



C3: Acceso seguro a la base de datos



C3: Acceso seguro a la base de datos

Vulnerabilidades evitadas

Inyección

Controles débiles del lado servidor

```
function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; 0 > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
      if (a) {
        (; 0 > i; i++)
          if (r = t.call(e[i], i, e[i]), r === !1) break
        (i in e)
          if (r = t.call(e[i], i, e[i]), r === !1) break;
        e
      }
      !b.call("\uffeff\u00a0") ? function(e) {
        null == e ? "" : b.call(e)
      } : n(e) {
        null == e ? "" : (e + "").replace(C, "")
      }
    }
  )
  function(e, t) {
    t || [];
    null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : n)
  }
  function(e, t, n) {
    (
      (a) return m.call(t, e, n);
      (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
      if (n in t && t[n] === e) return n
    )
  }
}
```


C4: Codificar y escapar datos

Codificar y escapar son técnicas defensivas destinadas a detener los ataques de inyección.

La codificación implica traducir caracteres especiales a una forma diferente pero equivalente que ya no es peligrosa en el intérprete de destino, por ejemplo, traducir el carácter `<` a la cadena `<` cuando se escribe en una página HTML.

Escapar implica agregar un carácter especial antes del carácter / cadena para evitar que se malinterprete, por ejemplo, agregar un carácter `\` antes de un carácter `"` (comillas dobles) para que se interprete como texto y no como cierre de cadena.



C4: Codificar y escapar datos

La codificación de salida

La codificación de salida se aplica mejor justo antes de que el contenido se pase al intérprete de destino. Si esta defensa se realiza demasiado pronto en el procesamiento de una solicitud, la codificación o el escape pueden interferir con el uso del contenido en otras partes del programa. Por ejemplo, si tu HTML escapa contenido antes de almacenar esos datos en la base de datos y la interfaz de usuario escapa automáticamente estos datos por segunda vez, el contenido no se mostrará correctamente debido a que tiene un doble escape.

C4: Codificar y escapar datos

La codificación de salida

Ejemplo Python. Django.

https://docs.djangoproject.com/es/3.0/_modules/django/utils/html/

```
"""HTML utilities suitable for global use."""

import html
import json
import re
from html.parser import HTMLParser
from urllib.parse import (
    parse_qs, quote, unquote, urlencode, urlsplit, urlunsplit,
)

from django.utils.encoding import punycode
from django.utils.functional import Promise, keep_lazy, keep_lazy_text
from django.utils.http import RFC3986_GENDELIMS, RFC3986_SUBDELIMS
from django.utils.safestring import SafeData, SafeString, mark_safe
from django.utils.text import normalize_newlines
```


C4: Codificar y Escapar datos

Vulnerabilidades evitadas

- Cross Site Scripting
- Injection

Herramientas

- [OWASP Java Encoder Project](#)
- [AntiXSSEncoder](#)

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
    } if (a) {
      (; o > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
      (i in e)
        if (r = t.call(e[i], i, e[i]), r === !1) break;
      e

      !b.call("\ufe0f\u00a0") ? function(e) {
        null == e ? "" : b.call(e)
      } n(e) {
        null == e ? "" : (e + "").replace(C, "")
      }

      function(e, t) {
        t || [];
        null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
        function(e, t, n) {
          (
            (a) return m.call(t, e, n);
            (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
            if (n in t && t[n] === e) return n

```

C5: Validar todas las entradas

La validación de entrada es una técnica de programación que garantiza que **solo los datos formateados correctamente puedan entrar a un componente del sistema de software.**



C5: Validar todas las entradas

Validez sintáctica y semántica

La **validez de sintaxis** significa que los datos están en la forma esperada. Por ejemplo, una aplicación puede permitir que un usuario seleccione un "ID de cuenta" de cuatro dígitos para realizar algún tipo de operación. La aplicación debe asumir que el usuario está ingresando una carga de inyección de SQL y debe verificar que los datos ingresados por el usuario tengan exactamente cuatro dígitos de longitud y consistan solo en números (además de utilizar la parametrización de consultas adecuada).

La **validez semántica** incluye solo aceptar entradas que estén dentro de un rango aceptable para la funcionalidad y el contexto de la aplicación. Por ejemplo, una fecha de inicio debe ser anterior a una fecha de finalización al elegir rangos de fechas.

C5: Validar todas las entradas

Lista blanca contra lista negra

- La validación de listas negras o listas negras intenta verificar que los datos proporcionados no contengan contenido "conocido incorrecto". Por ejemplo, una aplicación web puede bloquear la entrada que contiene el texto exacto <SCRIPT> para ayudar a prevenir XSS. Sin embargo, esta defensa podría eludirse con una etiqueta de secuencia de comandos en minúscula o una etiqueta de secuencia de comandos de mayúsculas y minúsculas.
- La validación de la lista blanca o lista blanca intenta comprobar que un dato dado coincide con un conjunto de reglas de "bien conocido". Por ejemplo, una regla de validación de lista blanca para un estado de EE. UU. Sería un código de 2 letras que es solo uno de los estados válidos de EE. UU.

C5: Validar todas las entradas

Lista blanca contra lista negra



Importante



Las listas blancas ayudan a limitar la superficie de ataque al garantizar que los datos tengan la validez sintáctica y semántica correcta, las listas negras ayudan a detectar y detener ataques potencialmente obvios.

C5: Validar todas las entradas

Validación del lado del cliente y del lado del servidor

La validación de entrada siempre debe realizarse en el lado del servidor por motivos de seguridad. Si bien la validación del lado del cliente puede ser útil tanto para fines funcionales como de seguridad, a menudo se puede omitir fácilmente. Esto hace que la validación del lado del servidor sea aún más fundamental para la seguridad. Por ejemplo, la validación de JavaScript puede alertar al usuario de que un campo en particular debe constar de números, pero la aplicación del lado del servidor debe validar que los datos enviados solo constan de números en el rango numérico apropiado para esa característica.



C5: Validar todas las entradas

Expresiones regulares

Las expresiones regulares ofrecen una forma de verificar si los datos coinciden con un patrón específico. Comencemos con un ejemplo básico.

La siguiente expresión regular se utiliza para definir una regla de lista blanca para validar nombres de usuario

```
^[a-z0-9_]{3,16}$
```

Esta expresión regular solo permite letras minúsculas, números y el carácter de subrayado. El nombre de usuario también está restringido a una longitud de 3 y 16 caracteres.

Precaución: potencial de denegación de servicio

Precaución: complejidad

C5: Validar todas las entradas

Límites de la validación de entrada

La validación de entrada no siempre hace que los datos sean “seguros” ya que ciertas formas de entrada compleja pueden ser “válidas” pero aún peligrosas. **Por ejemplo, una dirección de correo electrónico válida puede contener un ataque de inyección SQL o una URL válida puede contener un ataque Cross Site Scripting.** Siempre se deben aplicar defensas adicionales además de la validación de entrada a datos como la parametrización de consultas o el escape.

C5: Validar todas las entradas

Desafíos de la validación de datos serializados

El único patrón de arquitectura seguro es **no aceptar objetos serializados de fuentes no confiables o deserializar solo en una capacidad limitada solo para tipos de datos simples**. Debe evitar procesar formatos de datos serializados y usar formatos más fáciles de defender como JSON cuando sea posible.

Si eso no es posible, entonces se consideran una serie de defensas de validación al procesar datos serializados.

Implemente controles de integridad o encriptación de los objetos serializados

Aplique restricciones de tipo estrictas durante la deserialización antes de la creación del objeto

Aísle el código que se deserializa, de modo que se ejecute en entornos con muy pocos privilegios

Registre las excepciones y fallos de deserialización de seguridad

Restrinja o supervise la conectividad de red entrante y saliente de contenedores o servidores que deserializan

Supervise la deserialización, alertando si un usuario deserializa constantemente.

C5: Validar todas las entradas

Funcionalidad de validación en librerías y Frameworks

Todos los lenguajes y la mayoría de los **frameworks** proporcionan **librerías de validación o funciones las cuales deben ser aprovechadas para validar datos.**

Las librerías de validación generales cubren tipos de datos comunes, requisitos de longitud, rangos de números enteros, verificaciones "es nulo" y más. Muchas librerías y frameworks de validación te permiten definir tu propia expresión regular o lógica para la validación personalizada de una manera que le permite al programador aprovechar esa funcionalidad en toda su aplicación.

Form fields

`class Field(**kwargs)`

When you create a **Form** class, the most important part is defining the fields of the form. Each field has custom validation logic, along with a few other hooks.

Field.clean(value)

Although the primary way you'll use **Field** classes is in **Form** classes, you can also instantiate them and use them directly to get a better idea of how they work. Each **Field** instance has a **clean()** method, which takes a single argument and either raises a **django.core.exceptions.ValidationError** exception or returns the clean value:

```
>>> from django import forms
>>> f = forms.EmailField()
>>> f.clean('foo@example.com')
'foo@example.com'
>>> f.clean('invalid email address')
Traceback (most recent call last):
...
ValidationError: ['Enter a valid email address.']
```

django

The web framework for
perfectionists with deadlines.

C5: Validar todas las entradas

Vulnerabilidades evitadas

- La validación de entrada reduce la superficie de ataque de las aplicaciones y, a veces, puede dificultar los ataques contra una aplicación.
- La validación de entrada es una técnica que proporciona seguridad a determinadas formas de datos, específicas de determinados ataques y no se puede aplicar de forma fiable como regla de seguridad general.
- La validación de entrada no debe usarse como método principal para prevenir XSS, inyección SQL y otros ataques.

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
    } if (a) {
      (; o > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
      (i in e)
        if (r = t.call(e[i], i, e[i]), r === !1) break;
    } e

    !b.call("\uffeff\u00a0") ? function(e) {
      null == e ? "" : b.call(e)
    } n(e) {
      null == e ? "" : (e + "").replace(C, "")
    }

    function(e, t) {
      t || [];
      null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
      action(e, t, n) {
        (
          (a) return m.call(t, e, n);
          (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
          if (n in t && t[n] === e) return n

```

C5: Validar todas las entradas

Herramientas

- [OWASP Java HTML Sanitizer Project](#)
- [Java JSR-303/JSR-349 Bean Validation](#)
- [Java Hibernate Validator](#)
- [JEP-290 Filter Incoming Serialization Data](#)
- [Apache Commons Validator](#)
- [PHP's filter functions](#)

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
    }
    if (a) {
      (; o > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
      (i in e)
        if (r = t.call(e[i], i, e[i]), r === !1) break;
    }
    e

    !b.call("\uffeff\u00a0") ? function(e) {
      null == e ? "" : b.call(e)
    }
    n(e) {
      null == e ? "" : (e + "").replace(C, "")
    }

    function(e, t) {
      t || [];
      null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
      function(e, t, n) {
        (
          (a) return m.call(t, e, n);
          (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
          if (n in t && t[n] === e) return n

```


C6: Implementar la identidad digital

La identidad digital es la representación única de un usuario (u otro sujeto) mientras realiza una transacción en línea. La autenticación es el proceso de verificar que una persona o entidad es quien dice ser. La gestión de sesiones es un proceso mediante el cual un servidor mantiene el estado de la autenticación de los usuarios para que el usuario pueda continuar utilizando el sistema sin volver a autenticarse

La NIST Special Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management) proporciona una guía sólida sobre la implementación de controles de administración de sesión, autenticación e identidad digital.



C6: Implementar la identidad digital

Niveles de autenticación

NIST 800-63b describe **tres niveles** de garantía de autenticación denominados nivel de garantía de autenticación (AAL).

El **NIVEL I** de AAL está reservado para aplicaciones de menor riesgo que no contienen PII u otros datos privados. En el nivel 1 de AAL solo se requiere autenticación de factor único, generalmente mediante el uso de una contraseña.



C6: Implementar la identidad digital

Nivel I

Las contraseñas deben cumplir como mínimo los siguientes requisitos:

- Tener al menos 8 caracteres de longitud si también se utilizan la autenticación multifactor (MFA) y otros controles. Si MFA no es posible, debe aumentarse a al menos 10 caracteres.
- Todos los caracteres ASCII impresos, así como el carácter de espacio, deben ser aceptables en los secretos memorizados
- Fomentar el uso de contraseñas largas
- Eliminar los requisitos de complejidad, ya que se ha encontrado que tienen una eficacia limitada. En su lugar, se recomienda la adopción de MFA o contraseñas de mayor longitud.
- Asegúrese de que las contraseñas utilizadas no sean contraseñas de uso común que ya se hayan filtrado en un compromiso anterior. Puede optar por bloquear las 1000 o 10000 contraseñas más comunes que cumplen los requisitos de longitud anteriores y se encuentran en listas de contraseñas comprometidas. El siguiente enlace contiene las contraseñas más comunes:
<https://github.com/danielmiessler/SecLists/tree/master/Passwords>



Implementar un mecanismo de recuperación de contraseña segura

Implementar almacenamiento seguro de contraseñas

C6: Implementar la identidad digital

Nivel I

Ejemplo de hash seguro de contraseñas en python usando el algoritmo bcrypt.

```
#!/usr/bin/env python3

import bcrypt

passwd = b's$cret12'

salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)

print(salt)
print(hashed)
```

C6: Implementar la identidad digital

Nivel II

NIST 800-63b AAL nivel 2 está reservado para aplicaciones de mayor riesgo que contienen "PII (Información Personal Identificable) autoafirmada u otra información personal disponible en línea". En el nivel 2 de AAL, se requiere la autenticación multifactor, incluida la OTP (contraseña válida para una autenticación) u otras formas de implementación multifactor.

La autenticación multifactor (MFA) garantiza que los usuarios sean quienes dicen ser al exigirles que se identifiquen con una combinación de:

- Algo que sepa: contraseña o PIN
- Algo de su propiedad: una ficha o un teléfono
- Algo que eres: datos biométricos, como una huella digital

C6: Implementar la identidad digital

Nivel III

NIST 800-63b Authentication Assurance Level 3 (AAL3) se requiere cuando el impacto de los sistemas comprometidos podría conducir a daños personales, pérdidas financieras significativas, dañar el interés público o involucrar violaciones civiles o criminales. AAL3 requiere autenticación que se "basa en la prueba de posesión de una clave a través de un protocolo criptográfico". Este tipo de autenticación se utiliza para lograr el mayor nivel de garantía de autenticación. Esto se hace normalmente a través de módulos criptográficos de hardware.

C6: Implementar la identidad digital

Sesiones

Manejo de Sesiones

Una vez que ha tenido lugar la autenticación inicial exitosa del usuario, una aplicación puede optar por rastrear y mantener este estado de autenticación durante un período de tiempo limitado. Esto permitirá al usuario continuar usando la aplicación sin tener que volver a autenticarse con cada solicitud. El seguimiento de este estado de usuario se denomina Gestión de sesiones

Generación y caducidad de sesiones

El estado del usuario se rastrea en una sesión. Esta sesión normalmente se almacena en el servidor para la gestión de sesiones tradicional basada en web. A continuación, se le da al usuario un identificador de sesión para que pueda identificar qué sesión del lado del servidor contiene los datos de usuario correctos. El cliente solo necesita mantener este identificador de sesión, que también mantiene los datos confidenciales de la sesión del lado del servidor fuera del cliente.



C6: Implementar la identidad digital

Sesiones

Controles que se deben tener en cuenta al crear o implementar soluciones de administración de sesiones:

- Asegúrese de que la identificación de la sesión sea larga, única y aleatoria.
- La aplicación debe generar una nueva sesión o al menos rotar la identificación de la sesión durante la autenticación y reautenticación.
- La aplicación debe implementar un tiempo de espera inactivo después de un período de inactividad y una vida útil máxima absoluta para cada sesión, después del cual los usuarios deben volver a autenticarse. La duración de los tiempos de espera debe ser inversamente proporcional al valor de los datos protegidos.

Consulte la [hoja de referencia de gestión de sesiones para obtener más detalles](#). La Sección 3 de ASVS cubre los requisitos adicionales de gestión de sesiones.



C6: Implementar la identidad digital

Presta atención



La identidad digital, la autenticación y la gestión de sesiones son temas muy importantes. Estamos rascando la superficie del tema de la identidad digital aquí. Asegúrate de que tu talento sea capaz de mantener la complejidad involucrada con la mayoría de las soluciones de identidad.



C6: Implementar la identidad digital

Vulnerabilidades evitadas

- OWASP TOP 2. Manejo de sesiones y autenticación rota
- OWASP MOBILE. Autenticación insegura

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
      if (a) {
        (; o > i; i++)
          if (r = t.call(e[i], i, e[i]), r === !1) break
        (i in e)
          if (r = t.call(e[i], i, e[i]), r === !1) break;
        e
      }
    }
  )
  b.call("\uffeff\u00a0") ? function(e) {
    null == e ? "" : b.call(e)
  } : function(e) {
    null == e ? "" : (e + "").replace(C, "")
  }
}

function(e, t) {
  t || [];
  null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : n)
}

function(e, t, n) {
  (
    (a) return m.call(t, e, n);
    (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
    if (n in t && t[n] === e) return n
  )
}

```


C7: Aplicar controles de acceso

El control de acceso o autorización es el proceso de otorgar o denegar solicitudes específicas de un usuario, programa o proceso. El control de acceso también implica el acto de otorgar y revocar esos privilegios.

Cabe señalar que la autorización (verificar el acceso a recursos o funciones específicos) no es equivalente a la autenticación (verificar la identidad).



C7: Aplicar controles de acceso

Principios de diseño de control de acceso

1. Diseño de control de acceso desde el principio

2. Obliga a que todas las solicitudes pasen por controles de control de acceso

3. Denegar por defecto

4. Principio de privilegio mínimo

5. No “Hardcodear” roles

6. Registrar todos los eventos de control de acceso

C7: Aplicar controles de acceso

Vulnerabilidades evitadas

- OWASP TOP 5. Control de acceso roto
- OWASP MOBILE. Autenticación insegura.

Herramientas

- OWASP ZAP with the optional Access Control Testing add-on

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
    } if (a) {
      (; o > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
      (i in e)
        if (r = t.call(e[i], i, e[i]), r === !1) break;
      e

    |b.call("\uffeff\u00a0") ? function(e) {
      null == e ? "" : b.call(e)
    } n(e) {
      null == e ? "" : (e + "").replace(C, "")
    }

    function(e, t) {
      t || [];
      null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
    } function(e, t, n) {
      (
        (a) return m.call(t, e, n);
        (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
        if (n in t && t[n] === e) return n

```

C8: Proteja los datos en todas partes

Los datos sensibles tales como contraseñas, números de tarjetas de crédito, registros de salud, información personal y secretos empresariales requieren protección adicional, particularmente si esos datos están sujetos a las leyes de privacidad (Reglamento General de Protección de Datos de la UE, GDPR), reglas de protección de datos financieros como el Estándar de Seguridad de Datos de PCI (PCI DSS) u otras regulaciones.



C8: Proteja los datos en todas partes

A tener en cuenta:

Clasificación de los datos

Encriptación de datos en
tránsito

Encriptar datos en reposo

Aplicación móvil:
almacenamiento local
seguro

El ciclo de vida de las
claves

Gestión de secretos de
aplicaciones

C8: Proteja los datos en todas partes

Vulnerabilidades evitadas

- OWASP TOP 3. Datos sensibles expuestos.
- OWASP MOBILE.
Almacenamiento de datos inseguro.

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
      else
        for (i in e)
          if (r = t.apply(e[i], n), r === !1) break
      if (a) {
        (; o > i; i++)
          if (r = t.call(e[i], i, e[i]), r === !1) break
        (i in e)
          if (r = t.call(e[i], i, e[i]), r === !1) break;
        e
      }
    }
  )
  |b.call("\ufe0f\u00a0") ? function(e) {
    null == e ? "" : b.call(e)
  } : function(e) {
    null == e ? "" : (e + "").replace(C, "")
  }
}

function(e, t) {
  t || [];
  null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : n)
}

function(e, t, n) {
  (
    (a) return m.call(t, e, n);
    (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
    if (n in t && t[n] === e) return n
  )
}

```

C8: Proteja los datos en todas partes

Herramientas

- [SSLyze](#) - SSL configuration scanning library and CLI tool
- [SSL Labs](#) - Free service for scanning and checking TLS/SSL configuration
- [OWASP O-Saft TLS Tool](#) - TLS connection testing tool
- [GitRob](#) - Command line tool to find sensitive information in publicly available files on GitHub
- [TruffleHog](#) - Searches for secrets accidentally committed
- [KeyWhiz](#) - Secrets manager
- [Hashicorp Vault](#) - Secrets manager
- [Amazon KMS](#) - Manage keys on Amazon AWS

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; o > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
    }
    else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break
    if (a) {
      (; o > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
    }
    (i in e)
      if (r = t.call(e[i], i, e[i]), r === !1) break;
    e
  )
  |b.call("\uffeff\u00a0") ? function(e) {
    null == e ? "" : b.call(e)
  }
  n(e) {
    null == e ? "" : (e + "").replace(C, "")
  }
  function(e, t) {
    t || [];
    null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
  )
  function(e, t, n) {
    (
      (a) return m.call(t, e, n);
      (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
      if (n in t && t[n] === e) return n
    )
  }

```


C9: Implementar seguridad en logging y monitorización

El **logging** es un concepto que la mayoría de los desarrolladores ya utilizan con fines de depuración y diagnóstico. Securizar el logging es un concepto igualmente básico: para dar seguridad a la información de log durante las operaciones de ejecución de una aplicación.

El **monitoreo** es la revisión en vivo de la aplicación y la seguridad de los logs usando varias formas de automatización. Las mismas herramientas y patrones se pueden utilizar para operaciones, depuración y seguridad.



C9: Implementar seguridad en logging y monitorización

Logging:

Beneficios de seguridad en el logging

Implementación de seguridad en el logging

Logging para la detección y respuesta de intrusiones

Diseño seguro de logging

C9: Implementar seguridad en logging y monitorización

Vulnerabilidades evitadas

- Log Injection

Herramientas

Apache logging services

The Apache Logging Services Project creates and maintains open-source software related to the logging of application behavior and released at no charge to the public.



```
function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; 0 > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
    }
    else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break
    if (a) {
      (; 0 > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
    }
    (i in e)
      if (r = t.call(e[i], i, e[i]), r === !1) break;
    e
  )

  !b.call("\ufe0f\u00a0") ? function(e) {
    null == e ? "" : b.call(e)
  } : function(e) {
    null == e ? "" : (e + "").replace(C, "")
  }

  function(e, t) {
    t || [];
    null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : n)
  }

  function(e, t, n) {
    (
      (a) return m.call(t, e, n);
      (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
      if (n in t && t[n] === e) return n
    )
  }
}
```


C10. Manejar todos los errores y excepciones

El manejo de excepciones es un concepto de programación que permite que una aplicación responda a diferentes estados de error (como red inactiva, conexión de base de datos fallida, etc.) de varias formas. Manejar las excepciones y los errores correctamente es fundamental para que el código sea confiable y seguro.

El manejo de errores también es importante desde la perspectiva de la detección de intrusos. Ciertos ataques contra su aplicación pueden desencadenar errores que pueden ayudar a detectar ataques en curso.



C10. Manejar todos los errores y excepciones



Administre las excepciones de manera centralizada para evitar bloques try / catch duplicados en el código. Asegúrese de que todo comportamiento inesperado se maneje correctamente dentro de la aplicación.



Asegúrese de que los mensajes de error que se muestran a los usuarios no filtran datos críticos, pero siguen siendo lo suficientemente detallados para permitir la respuesta adecuada del usuario.



Asegúrese de que las excepciones se registren de manera que brinden suficiente información para que los equipos de soporte, control de calidad, forenses o de respuesta a incidentes comprendan el problema.



Pruebe y verifique cuidadosamente el código de manejo de errores

C10: Manejar todos los errores y excepciones

Vulnerabilidades evitadas

- CWE 209 . Exposición de información a través de mensajes de error

```

function(e, t, n) {
  i = 0,
  e.length,
  M(e);
  (
    (a) {
      for (; 0 > i; i++)
        if (r = t.apply(e[i], n), r === !1) break
    }
    else
      for (i in e)
        if (r = t.apply(e[i], n), r === !1) break
    if (a) {
      (; 0 > i; i++)
        if (r = t.call(e[i], i, e[i]), r === !1) break
    }
    (i in e)
      if (r = t.call(e[i], i, e[i]), r === !1) break;
    e

    |b.call("\uffeff\u00a0") ? function(e) {
      null == e ? "" : b.call(e)
    }
    n(e) {
      null == e ? "" : (e + "").replace(C, "")
    }

    function(e, t) {
      t || [];
      null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e]
    )
    function(e, t, n) {
      (
        (a) return m.call(t, e, n);
        (r = t.length, n = n ? 0 > n ? Math.max(0, r + n) : n : 0; r > n;
        if (n in t && t[n] === e) return n

```




CONCLUSIONES

1

Aplicación de controles proactivos en desarrollo

2

Uso de herramientas para aplicación de controles proactivos

3

Aplicación de controles proactivos para solucionar problemas del OWASP TOP 10

MUCHAS GRACIAS POR SU ATENCIÓN



dtinedo@grupomainjobs.com



Diego Tinedo Rodríguez
[linkedin.com/in/diego-tinedo](https://www.linkedin.com/in/diego-tinedo)



twitter.com/eiposgrados



facebook.com/eiposgrados



instagram.com/eiposgrados